1차시 과제 풀이

문제 목록

문제 번호	문제 이름	출처
BOJ 2751	수 정렬하기 2	
BOJ 11650	좌표 정렬하기	
BOJ 10825	국영수	
BOJ 2167	2차원 배열의 합	
BOJ 1920	수 찾기	
BOJ 2805	나무 자르기	COCI 2011/2012 #5
BOJ 2512	예산	KOI 2012 고등부
BOJ 7453	합이 0인 네 정수	SWERC 2005
BOJ 1637	날카로운 눈	SEERC 2007
BOJ 10227	삶의 질	IOI 2010 Day1
BOJ 17298	오큰수	
BOJ 6549	히스토그램에서 가장 큰 직사각형	
BOJ 2867	수열의 값	COCI 2010/2011 #3
BOJ 8986	전봇대	KOI 2013 고등부
BOJ 13310	먼 별	KOI 2016 고등부
BOJ 15977	조화로운 행렬	KOI 2018 고등부
BOJ 20090	Aliens	IOI 2016 Day2

BOJ 2751 수 정렬하기 2

`std::sort` 사용법 연습

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<int> V(N);
    for(auto &i : V) cin >> i;
    sort(V.begin(), V.end());
    for(auto i : V) cout << i << "\n";
}
```

매크로(`#define`)을 사용하면 더 편하게 할 수 있다.

```
#define all(v) v.begin(), v.end()

sort(all(V)); //sort(V.begin(), V.end());
```

BOJ 11650 좌표 정렬하기

`std::sort`와 `std::pair` 연습 문제

```
#include <bits/stdc++.h>
#define x first
#define y second
#define all(v) v.begin(), v.end()
using namespace std;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<pair<int, int>> V(N);
    for(auto &i : V) cin >> i.x >> i.y;
    sort(all(V));
    for(auto i : V) cout << i.x << " " << i.y << "\n";
}
```

BOJ 10825 국영수

`std::sort` 연습 문제

```
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
using namespace std;

struct Info{
    int kor, eng, mat;
    string name;
};

bool comp(const Info &a, const Info &b){
    if(a.kor != b.kor) return a.kor > b.kor;
    if(a.eng != b.eng) return a.eng < b.eng;
    if(a.mat != b.mat) return a.mat > b.mat;
    return a.name < b.name;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<int> V(N);
    for(auto &i : V) cin >> i.name >> i.kor >> i.eng >> i.mat;
    sort(all(V), comp);
    for(const auto &i : V) cout << i.name << "\n";
}
```

BOJ 2167 2차원 배열의 합

$S[i][j] = \sum_{y=1}^{i}\sum_{x=1}^{j} A[y][x]$라고 정의하자. S는 $O(NM)$ 시간에 전처리할 수 있다. (아래 코드 참고)

$\sum_{y=r_1}^{r_2}\sum_{x=c_1}^{c_2} A[y][x]$는 $S[r_2][c_2] - S[r_2][c_1-1] - S[r_1-1][c_2] + S[r_1-1][c_1-1]$과 동일하므로 $O(1)$ 시간에 계산할 수 있다. 왜 되는지 궁금하면 그림을 그려보자.

```
#include <bits/stdc++.h>
using namespace std;

int N, M, K;
int A[333][333], S[333][333];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) cin >> A[i][j];
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) {
        S[i][j] = S[i-1][j] + S[i][j-1] - S[i-1][j-1] + A[i][j];
    }
    cin >> K;
    while(K--){
        int r1, c1, r2, c2; cin >> r1 >> c1 >> r2 >> c2;
        cout << S[r2][c2] - S[r1-1][c2] - S[r2][c1-1] + S[r1-1][c1-1] << "\n";
    }
}
```

BOJ 1920 수 찾기

이분 탐색 연습 문제

```
#include <bits/stdc++.h>
using namespace std;

int N, Q, A[101010];

bool search(int v){
    int l = 1, r = N;
    while(l <= r){
        int m = l + r >> 1;
        if(A[m] == v) return true;
        else if(A[m] < v) l = m + 1;
        else r = m - 1;
    }
    return false;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    sort(A+1, A+N+1);
    cin >> M;
```

```
    while(M--){
        int t; cin >> t;
        cout << search(t) << "\n";
    }
}
```

BOJ 2805 나무 자르기

Parametric Search 연습 문제

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, K, A[1010101];

bool f(ll v){
    ll sum = 0;
    for(int i=1; i<=N; i++){
        if(A[i] > v) sum += A[i] - v;
    }
    return sum >= K;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1; i<=N; i++) cin >> A[i];
    ll l = 0, r = 2e9;
    while(l < r){
        ll m = l + r + 1 >> 1;
        if(f(m)) l = m;
        else r = m - 1;
    }
    cout << l;
}
```

BOJ 2512 예산

Parametric Search 연습 문제

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, K, A[10101];

bool f(ll v){
    ll sum = 0;
    for(int i=1; i<=N; i++){
        sum += min(A[i], v);
    }
    return sum <= K;
}

int main(){
```

```
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    cin >> K;
    if(accumulate(A+1, A+N+1, 0LL) <= K){
        cout << *max_element(A+1, A+N+1);
        return 0;
    }
    ll l = 0, r = 2e9;
    while(l < r){
        ll m = l + r + 1 >> 1;
        if(f(m)) l = m;
        else r = m - 1;
    }
    cout << l;
}
```

BOJ 7453 합이 0인 네 정수

배열이 2개일 때는 어떻게 해결할 수 있을까? 배열 A의 각 원소 $A[i]$에 대해, 배열 B에 있는 $-A[i]$의 개수를 세어주면 된다. 이분 탐색을 이용하면 시간 복잡도는 $O(N \log N)$이 된다.

배열이 4개일 때도 비슷한 방식으로 해결할 수 있다. 배열 A의 원소와 B의 원소를 이용해 나타낼 수 있는 모든 경우의 수를 배열 AB에 저장하자. 마찬가지로 CD도 전처리하자. 각 배열의 크기는 정확히 N^2이다. 이 두 배열에 대해 배열이 2개일 때의 풀이를 적용하면 $O(N^2 \log N^2) = O(N^2 \log N)$에 문제를 해결할 수 있다.

`std::equal_range`를 이용하면 쉽게 구현할 수 있다.

```
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
using namespace std;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<int> A(N), B(N), C(N), D(N), AB, CD;
    for(int i=0; i<N; i++) cin >> A[i] >> B[i] >> C[i] >> D[i];
    AB.reserve(N*N);
    CD.reserve(N*N);
    for(int i=0; i<N; i++) for(int j=0; j<N; j++)
        AB.push_back(A[i] + B[j]), CD.push_back(-(C[i] + D[j]));
    sort(all(AB));
    sort(all(CD));

    long long ans = 0;
    for(auto i : AB){
        auto range = equal_range(all(CD), i);
        ans += distance(range.x, range.y);
    }
    cout << ans;
}
```

BOJ 1637 날카로운 눈

홀수 개 짜리 원소가 존재한다면, **수의 개수**에 대한 누적합이 짝수에서 홀수로 변하는 지점이 **정확히 한 곳** 존재한다. 홀수 개 짜리 원소가 없다면 누적합은 항상 짝수가 된다.

처음으로 홀수가 되는 지점을 파라메트릭 서치로 구해주면 된다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, A[20202], B[20202], C[20202];

ll f(ll x){
    ll sum = 0;
    for(int i=1; i<=N; i++){
        if(x >= A[i]) sum += (min(x, B[i]) - A[i]) / C[i] + 1;
    }
    return sum;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i] >> B[i] >> C[i];

    ll l = 0, r = 1LL << 32;
    while(l < r){
        ll m = l + r >> 1;
        if(f(m) & 1) r = m;
        else l = m + 1;
    }
    if(f(r) & 1) cout << r << " " << f(r) - f(r-1);
    else cout << "NOTHING";
}
```

BOJ 10227 삶의 질

최솟값을 구하는 문제이므로 파라메트릭 서치를 생각해볼 수 있다. **중앙값이 X 이하인 직사각형이 존재하는지 판단**하는 문제를 해결하자.

주어진 이차원 배열 A에서, X보다 큰 원소는 1, X보다 작은 원소는 -1, X는 0으로 바꾼 새로운 배열 T를 생각해보자.

- 만약 직사각형의 중앙값이 정확히 X라면 배열 T에서 직사각형 영역의 합은 0이다.
- 중앙값이 X보다 작다면 직사각형 영역의 합은 음수이다.
- 중앙값이 X보다 크다면 직사각형 영역의 합은 양수이다.

직사각형 영역의 합을 구하는 것은 2D Prefix Sum을 이용해 $O(RC)$ 전처리를 하면 $O(1)$에 구할 수 있으므로 $O(RC \log RC)$에 문제를 해결할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;

int N, M, H, W, A[3030][3030], T[3030][3030];

void Init(int X){
```

```
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) {
        if(A[i][j] > X) T[i][j] = 1;
        else if(A[i][j] < X) T[i][j] = -1;
        else T[i][j] = 0;
    }
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) {
        T[i][j] += T[i-1][j] + T[i][j-1] - T[i-1][j-1];
    }
}

int Query(int r1, int r2, int c1, int c2){
    return T[r2][c2] - T[r2][c1-1] - T[r1-1][c2] + T[r1-1][c1-1];
}

bool f(int X){
    Init(X);
    for(int i=1; i<=N; i++) if(i+H-1 <= N) {
        for(int j=1; j<=M; j++) if(j+W-1 <= M) {
            if(Query(i, i+H-1, j, j+W-1) <= 0) return true;
        }
    }
    return false;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> H >> W;
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) cin >> A[i][j];
    int l = 0, r = 3030*3030;
    while(l < r){
        int m = l + r >> 1;
        if(f(m)) r = m;
        else l = m + 1;
    }
    cout << r;
}
```

BOJ 17298 오큰수

Monotone Stack 연습 문제

```
#include <bits/stdc++.h>
#define all(v) v.begin(), v.end()
using namespace std;
constexpr int INF = 0x3f3f3f3f;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<int> V(N), R(N);
    for(auto &i : V) cin >> i;

    stack<int, vector<int>> stk; stk.push(INF);
    for(int i=N-1; ~i; i--){
        while(stk.top() <= V[i]) stk.pop();
        R[i] = stk.top() == INF ? -1 : stk.top();
        stk.push(V[i]);
```

```
    }
    for(auto i : R) cout << i << " ";
}
```

BOJ 6549 히스토그램에서 가장 큰 직사각형

직사각형의 높이는 입력으로 주어진 h_i들만 봐도 충분하다.

i번째 막대의 높이 h_i를 온전히 포함하는(높이가 h_i 이상인) 직사각형의 최대 너비를 $T(N)$ 시간에 구할 수 있다면, (최대 너비)·h_i의 최댓값을 구하면 되므로 $O(N \cdot T(N))$ 시간에 문제를 해결할 수 있다. h_i를 온전히 포함하는 직사각형의 최대 너비를 구하는 방법을 알아보자.

i번째 막대의 왼쪽에 있는 막대 중 처음으로 h_i보다 작은 막대의 위치, 오른쪽에 있는 막대 중 처음으로 h_i보다 작은 막대의 위치를 구하면 된다. 이는 Monotone Stack으로 각각 amortized $O(1)$에 구할 수 있으므로 전체 문제를 $O(N)$에 해결할 수 있다.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using PII = pair<int, int>;

ll N, A[101010], L[101010], R[101010];

void Solve(){
    for(int i=1; i<=N; i++) cin >> A[i];

    stack<PII, vector<PII>> stk;
    stk.emplace(-1, 0);
    for(int i=1; i<=N; i++){
        while(stk.top().x >= A[i]) stk.pop();
        L[i] = stk.top().y + 1;
        stk.emplace(A[i], i);
    }
    while(stk.size()) stk.pop();

    stk.emplace(-1, N+1);
    for(int i=N; i>=1; i--){
        while(stk.top().x >= A[i]) stk.pop();
        R[i] = stk.top().y - 1;
        stk.emplace(A[i], i);
    }

    ll ans = 0;
    for(int i=1; i<=N; i++){
        ans = max(ans, A[i] * (R[i] - L[i] + 1));
    }
    cout << ans << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    while(true){
        cin >> N; if(!N) break;
        Solve();
    }
```

```
}
```

BOJ 2867 수열의 값

해설의 편의를 위해 입력으로 들어오는 모든 수가 서로 다르다고 하자.

어떤 수 $A[i]$가 최솟값인 구간 $[Lmin[i], Rmin[i]]$와 최댓값인 구간 $[Lmax[i], Rmax[i]]$라고 하면, $A[i]$가 최솟값인 구간은 $(i - Lmin[i] + 1) \cdot (Rmin[i] - i + 1)$가지, 최댓값인 구간은 $(i - Lmax[i] + 1) \cdot (Rmax[i] - i + 1)$가지이다.

$Lmin, Rmin, Lmax, Rmax$는 Monotone Stack을 이용해 각각 $O(N)$에 구할 수 있으므로 $O(N)$에 문제를 해결할 수 있다.

만약 중복된 수가 주어진다면 왼쪽 경계는 같은 수를 허용하고 오른쪽 경계는 같은 수를 허용하지 않도록 처리하면 된다.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using PII = pair<int, int>;
constexpr int INF = 0x3f3f3f3f;

ll N, A[303030], Lmin[303030], Lmax[303030], Rmin[303030], Rmax[303030];
stack<PII> Smin, Smax;

inline void clear(){
    while(Smin.size()) Smin.pop();
    while(Smax.size()) Smax.pop();
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];

    Smin.emplace(0, 0);
    Smax.emplace(INF, 0);
    for(int i=1; i<=N; i++){
        while(Smin.size() && Smin.top().x >= A[i]) Smin.pop();
        while(Smax.size() && Smax.top().x <= A[i]) Smax.pop();
        Lmin[i] = Smin.top().y + 1;
        Lmax[i] = Smax.top().y + 1;
        Smin.emplace(A[i], i);
        Smax.emplace(A[i], i);
    }

    clear();

    Smin.emplace(0, N+1);
    Smax.emplace(INF, N+1);
    for(int i=N; i>=1; i--){
        while(Smin.size() && Smin.top().x > A[i]) Smin.pop();
        while(Smax.size() && Smax.top().x < A[i]) Smax.pop();
        Rmin[i] = Smin.top().y - 1;
        Rmax[i] = Smax.top().y - 1;
```

```
        Smin.emplace(A[i], i);
        Smax.emplace(A[i], i);
    }

    ll ans = 0;
    for(int i=1; i<=N; i++){
        ll minCount = (Rmin[i] - i + 1) * (i - Lmin[i] + 1);
        ll maxCount = (Rmax[i] - i + 1) * (i - Lmax[i] + 1);
        ans += A[i] * (maxCount - minCount);
    }
    cout << ans;
}
```

BOJ 8986 전봇대

모든 점을 평행이동해서 $x_0 = 0$으로 만들어도 정답은 동일하다. $x_0 = 0$이라고 가정하면, x_i를 $i \cdot m$으로 옮길 때 이동 거리 합의 최솟값을 찾는 문제다. 다시 말해, $f(m) = \sum_{i=0}^{N-1} |x_i - im|$의 최솟값을 찾는 문제다.

시그마 안에 있는 각 항은 unimodal하다. unimodal 함수의 합도 unimodal하므로 $f(m)$은 unimodal하다. 삼분 탐색을 통해 최솟값을 찾을 수 있다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, A[101010];

ll f(ll m){
    ll ret = 0;
    for(int i=0; i<N; i++) ret += abs(A[i] - i*m);
    return ret;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=0; i<N; i++) cin >> A[i];
    for(int i=N-1; ~i; i--) A[i] -= A[0];
    A[0] = 0;

    ll l = 0, r = 1e9;
    while(l+3 <= r){
        ll m1 = (l + l + r) / 3, m2 = (l + r + r) / 3;
        if(f(m1) < f(m2)) r = m2;
        else l = m1;
    }
    ll ans = 0x3f3f3f3f3f3f3f3f;
    for(ll i=l; i<=r; i++) ans = min(ans, f(i));
    cout << ans;
}
```

BOJ 13310 먼 별

$N = 2$인 경우를 생각해보자. 시간 t에 대한 점의 거리를 함수 $f(t)$를 생각해보면 $f(t)$는 unimodal하다는 것을 할 수 있다. $N > 2$일 때는 unimodal한 함수끼리 max를 취한 것이기 때문에 여전히 unimodal하므로 삼분 탐색을 적용할 수 있다.

N개의 점이 주어졌을 때 가장 먼 두 점은 Convex Hull과 Rotating Calipers를 이용해 $O(N \log N)$에 구할 수 있다. 그러므로 문제를 $O(N \log N \log T)$에 해결할 수 있다.

```
#include <bits/stdc++.h>
#define x first
#define y second
#define all(v) v.begin(), v.end()
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;
constexpr Point O = {0, 0};

struct Star{
    ll x, y, dx, dy;
    Point get(ll t) const { return {x + dx*t, y + dy*t}; }
};

int CCW(const Point &p1, const Point &p2, const Point &p3){
    ll res = (p2.x-p1.x)*(p3.y-p2.y) - (p3.x-p2.x)*(p2.y-p1.y);
    return (res > 0) - (res < 0);
}
ll D2(const Point &p1, const Point &p2){
    return (p2.x-p1.x)*(p2.x-p1.x) + (p2.y-p1.y)*(p2.y-p1.y);
}

vector<Point> ConvexHull(vector<Point> &V){
    vector<Point> H;
    swap(V[0], *min_element(all(V)));
    sort(1+all(V), [&](const Point &a, const Point &b){
        if(int cw = CCW(V[0], a, b); cw) return cw > 0;
        return D2(V[0], a) < D2(V[0], b);
    });
    for(const auto &i : V){
        while(H.size() >= 2 && CCW(H[H.size()-2], H.back(), i) <= 0)
H.pop_back();
        H.push_back(i);
    }
    return move(H);
}

pair<Point, Point> RotatingCalipers(const vector<Point> &H){
    auto Check = [&](Point s1, Point e1, Point s2, Point e2){
        Point p1 = {e1.x-s1.x, e1.y-s1.y};
        Point p2 = {e2.x-s2.x, e2.y-s2.y};
        return CCW(O, p1, p2) >= 0;
    };
    ll mx = 0; Point a, b;
    for(int i=0, j=0; i<H.size(); i++){
        while(j+1 < H.size() && Check(H[i], H[i+1], H[j], H[j+1])){
            if(ll now = D2(H[i], H[j]); mx < now) mx = now, a = H[i], b = H[j];
            j++;
```

```
        }
        if(ll now = D2(H[i], H[j]); mx < now) mx = now, a = H[i], b = H[j];
    }
    return {a, b};
}

ll N, T;
vector<Star> V;

ll f(ll t){
    vector<Point> P; P.reserve(N);
    for(const auto &i : V) P.push_back(i.get(t));
    auto [a,b] = RotatingCalipers(ConvexHull(P));
    return D2(a, b);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> T; V.resize(N);
    for(auto &i : V) cin >> i.x >> i.y >> i.dx >> i.dy;

    ll l = 0, r = T;
    while(l+3 <= r){
        ll m1 = (l + l + r) / 3, m2 = (l + r + r) / 3;
        if(f(m1) > f(m2)) l = m1;
        else r = m2;
    }

    ll mn = 0x3f3f3f3f3f3f3f3f, idx = l;
    for(ll i=l; i<=r; i++){
        if(ll now = f(i); mn > now) mn = now, idx = i;
    }
    cout << idx << "\n" << mn;
}
```

BOJ 15977 조화로운 행렬

각 열을 첫째 행을 기준으로 정렬하자. $M = 2$이면 둘째 행의 LIS를 구하는 문제가 되고, $M = 3$이면 (둘째 행, 셋째 행) pair에 대한 LIS를 구하는 문제가 된다. $M = 2$는 $M = 3$으로 바꿀 수 있으므로 pair에 대한 LIS를 구하는 방법만 알면 된다.

LIS를 세그먼트 트리로 구할 수 있듯이, pair에 대한 LIS도 2D 세그먼트 트리를 이용하면 $O(N \log^2 N)$에 구할 수 있지만, 더 좋은 풀이가 있으므로 그 풀이를 알아보자.

$lis[i]$를 i번째 원소로 끝나는 가장 긴 증가 부분 수열의 길이라고 정의하자. 어떤 K에 대해 $lis[i] = K$를 만족하는 i를 모두 모아서 보면, $B[i]$가 증가할수록 항상 $C[i]$가 감소한다는 것을 알 수 있다. (귀류법을 쓰면 간단하게 증명할 수 있다.)

$B[i]$가 증가함에 따라 $C[i]$가 감소한다는 것은, 2차원 평면에 $(B[i], C[i])$ 점을 찍었을 때 우하향 계단 형태가 나온다는 것을 의미한다. 이러한 점들은 `std::set / std::map`을 이용해 관리해줄 수 있다. LIS는 최대 20만이므로 `std::set`을 20만개 관리하면 된다.

$lis[i] = K$가 되기 위해서는 $j < i \wedge lis[j] = K - 1$을 만족하는 j가 존재해야 한다. 그러므로 i를 1부터 N까지 차례대로 보면서 Parametric Search를 하면 $lis[i]$를 구할 수 있다.

정답은 $\max\{lis[i]\}$가 되고, $O(N \log^2 N)$에 구할 수 있다.

```cpp
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using Point = pair<int, int>;

struct Triplet{
    int x, y, z;
    bool operator < (const Triplet &t) const {
        return tie(x, y, z) < tie(t.x, t.y, t.z);
    }
};

struct Envelope{
    set<Point> st;
    void insert(const Point &pt){
        auto it = next(st.insert(pt).first);
        while(it != st.end() && it->y >= pt.y) it = st.erase(it);
    }
    bool Query(const Point &pt){
        auto it = st.lower_bound(pt);
        return it != st.begin() && prev(it)->y <= pt.y;
    }
};

int K, N, LIS[202020];
Triplet Input[202020];
Point A[202020];
Envelope st[202020];

int Get(int i){
    int l = 1, r = N;
    while(l < r){
        int m = l + r + 1 >> 1;
        if(st[m-1].Query(A[i])) l = m;
        else r = m - 1;
    }
    return l;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> K >> N;
    for(int i=1; i<=N; i++) cin >> Input[i].x;
    for(int i=1; i<=N; i++) cin >> Input[i].y;
    if(K == 3) for(int i=1; i<=N; i++) cin >> Input[i].z;
    else for(int i=1; i<=N; i++) Input[i].z = Input[i].y;

    sort(Input+1, Input+N+1);
    for(int i=1; i<=N; i++) A[i] = {Input[i].y, Input[i].z};

    for(int i=1; i<=N; i++){
        LIS[i] = Get(i);
        st[LIS[i]].insert(A[i]);
    }
    cout << *max_element(LIS+1, LIS+N+1);
}
```

BOJ 20090 Aliens

Aliens Trick 연습 문제

```
#include "aliens.h"
#include <bits/stdc++.h>
#define x first
#define y second
#define all(v) v.begin(), v.end()
using namespace std;
using ll = long long;
using PLL = pair<ll, ll>;
constexpr ll INF = 0x3f3f3f3f3f3f3f3f;

inline ll sq(ll v){ return v*v; }

struct Line{
    ll a, b, i;
    Line() : Line(0, INF, 0) {}
    Line(ll a, ll b, ll i) : a(a), b(b), i(i) {}
    ll f(ll x) const { return a * x + b; }
};

struct CHT{
    Line v[101010]; int pv, top;
    void clear(){ pv = top = 0; }
    int __cross(const Line &a, const Line &b, const Line &c){
        return (a.b - b.b) * (b.a - c.a) <= (c.b - b.b) * (b.a - a.a);
    }
    void update(Line l){
        while(top >= pv+2 && __cross(v[top-2], v[top-1], l)) top--;
        v[top++] = l;
    }
    PLL query(ll x){
        while(pv+1 < top && v[pv].f(x) >= v[pv+1].f(x)) pv++;
        return {v[pv].f(x), v[pv].i};
    }
} cht;

int N, K;
PLL A[101010];
ll D[101010], C[101010];

void init(int _n, int _m, int _k, const vector<int> &_r, const vector<int> &_c){
    K = _k;
    vector<PLL> pts;
    for(int i=0; i<_n; i++) pts.emplace_back(min(_r[i], _c[i]), max(_r[i],
_c[i]));
    sort(all(pts), [&](const PLL &p1, const PLL &p2){
        return p1.x != p2.x ? p1.x < p2.x : p1.y > p2.y;
    });
    for(const auto &i : pts) if(!N || A[N].y < i.y) A[++N] = i;
}

Line makeLine(int i){
    ll a = -2*A[i+1].x, b = D[i] + sq(A[i+1].x) - 2*A[i+1].x;
    if(i) b -= sq(max(0LL, A[i].y-A[i+1].x+1));
```

```
    return {a, b, i};
}

ll get(ll c){
    cht.clear();
    cht.update(makeLine(0));
    for(int i=1; i<=N; i++){
        auto res = cht.query(A[i].y);
        D[i] = res.x + sq(A[i].y+1) + c;
        C[i] = C[res.y] + 1;
        cht.update(makeLine(i));
    }
    return C[N];
}

ll take_photos(int _n, int _m, int _k, vector<int> _r, vector<int> _c){
    init(_n, _m, _k, _r, _c);
    K = min(N, K);

    ll l = 0, r = 1e15, ans = 0;
    while(l <= r){
        ll m = l + r >> 1, cnt = get(m);
        if(cnt == K) return D[N] - K*m;
        else if(cnt < K) r = m - 1;
        else l = m + 1, ans = max(ans, D[N] - K*m);
    }
    return ans;
}
```